AMATH 582 Homework 5

William Livernois (willll@uw.edu)

March 17, 2020

Abstract

Dynamic mode decomposition (DMD) was used on a video to track foreground and background objects between each frame. This algorithm was applied by creating the background from stationary modes and taking the difference from the original image to isolate moving features.

1 Introduction and Overview

Using singular value decomposition (SVD) on a set of flattened images it is possible to find a set of eigenimages that can be used to compress the system into a low-rank system of principle components. The DMD algorithm applies this idea to a video with regular time intervals, applying the constraint that there is a linear transformation of modes between each frame. This approximation can be used to isolate features that are relatively stationary between frames, which represents the background of the image.

The DMD algorithm for background isolation was generated and applied to two videos, one of a racetrack and one of a skier, each with a clear background and moving objects. The foreground objects were extracted as a difference between the original image and background at each time step and written to a video file.

2 Theoretical Background

The goal of DMD analysis is to constrain a system to modes that can be represented by a linear transformation (called the Koopman operator) between each time step. Given measurement matrix given by X_1 (columns with measurements from times 1 to N-1) and a second measurement matrix X_2 that is shifted by one time step (columns with measurements from time 2 to N), we can define the Koopman operator, A, by the relationship

$$X_2 = AX_1 \tag{1}$$

From this formula, A can be calculated using the inverse of the second matrix (called "exact" DMD) but this involves high-dimensional matrix with limited constraints, resulting A a matrix that is over-fitted to the measurements. Therefore, for the DMD algorithm SVD was used to reduce the rank of the system as an approximation of the initial measurements such that $X_1 \approx \tilde{X}_1 = U_r \Sigma_r V_r^*$ where he r represents the principle mode truncation. This can be plugged into Equation 1 to get

$$A \approx X_2 \tilde{X}_1^{-1} = X_2 (U_r \Sigma_r V_r^*)^{-1} = X_2 V_r \Sigma_r^{-1} U_r^* \tag{2}$$

Finally, this matrix can be converted to the low-rank basis using the U_r matrix to get

$$\tilde{A} = U_r^* A U_r = U_r^* X_2 V_r \Sigma_r^{-1} \tag{3}$$

Using this reduced Koopman operator, finding the dynamic modes becomes an eigenvalue problem solving $\tilde{A}\psi_k = \lambda \psi_k$. Combining these modes with the definition of the A in Equation 1, the time evolution of these ψ_k modes are given by

$$\psi_{k,n} = \tilde{A}^n \psi_{k,0} = (\lambda_k)^n \psi_{k,0} = \exp(\omega_k t) \psi_{k,0} \tag{4}$$

where $\omega_k = \log(\lambda_k)/\Delta t$ and $t = n\Delta t$. By definition, these modes form a complete basis describing the system, with the matrix of eigenvectors given as Ψ so that the time evolution of the whole system can be given by

$$\vec{x}(t) = \sum_k b_k \psi_k \exp(\omega_k t) = \Psi \cdot \vec{b} \exp(I\vec{\omega}t) \tag{5}$$

where $\vec{b}$ is the initial conditions of the system given by $\vec{b} = \Psi^{-1} \cdot \vec{x}(0)$ which represents a transform of the initial measurements to the eigenbasis[1]. The ω values of each mode are complex values, indicating exponential growth or periodic motion of the modes. For this application, modes with $\omega \approx 0$ were selected representing stationary background modes.

3 Algorithm Implementation and Development

The singular values and transformation matrices used in Equation 3 were extracted using `svd()` MATLAB function that uses a proprietary algorithm that is based on the `DGESVD` function in the LAPACK library[2]. The eigenvalues of the Koopman operator were determined using the built in `eig` MATLAB function. The generalized `DMDBgSub` function was created to extract the background modes, with a rank cutoff and $|\omega|$ cutoff value input variable.

The original video was converted to grayscale for the DMD processing. Time was counted by frame number, meaning that ΔT was set to 1 and time n corresponded to frame n. The foreground video was extracted by taking the difference between the the original image matrix and the background image at each frame. To account for any negative values from this difference the minimum pixel value was subtracted and then the frame was re-scaled to map back to the original grayscale range. In this implementation two videos (`monte_carlo_low.mp4` and `ski_drop_low.mp4`) were imported, processed, and divided into two output videos (foreground and background for each).

4 Computational Results

The reduced rank DMD used the first 100 SVD modes, which provided an accurate representation of the background and allowed the ω values to all exist in the left half plane as shown in Figure 1. There was a cluster of points located near the origin in each case, indicating a high number of stationary modes. A cutoff for $|\omega|$ was decided based on the visual results from the foreground removal. A comparison between two cutoffs values are shown in Figure 2, with the value of 0.2 being used for the final results. As can be seen from the video, a lower cutoff omits more of the car features from the background at the cost of removing some of the image stabilization from the higher amplitude modes. In this case, the "UBS" sign can be seen quite clearly behind the cars with the lower cutoff whereas it's almost completely gone at the higher cutoff.

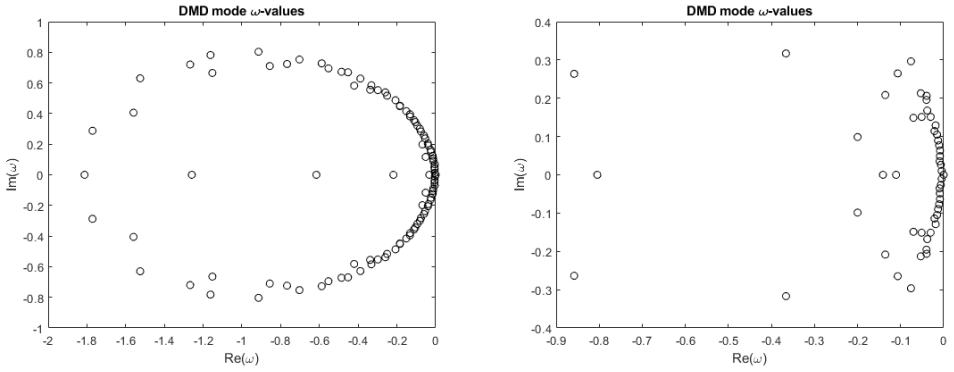

Figure 1: Plotting the complex ω values of the DMD modes from the racecar video (left) and skiing video (right)

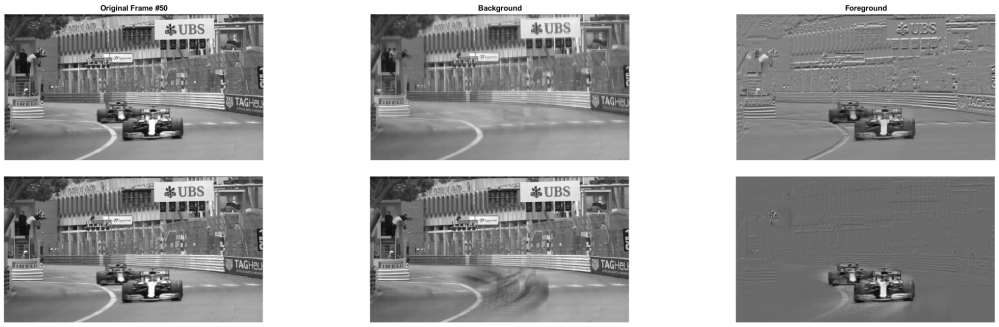

Figure 2: A comparison between two ω cutoffs for background removal: (top) $|\omega| < 0.01$ and (bottom) $|\omega| < 0.2$

Too high of a cutoff lead to capturing more of the cars in the background, ultimately creating less contrast in the foreground video.

Three frames from the racecar video (Figure) and the skiing video (Figure) are shown below. The original frames are compared to the decoupled background and foreground images from the procedure. A rank of 50 was chosen for the ski video (which had less overall variance) while the rank of 100 was used for the racecar video.

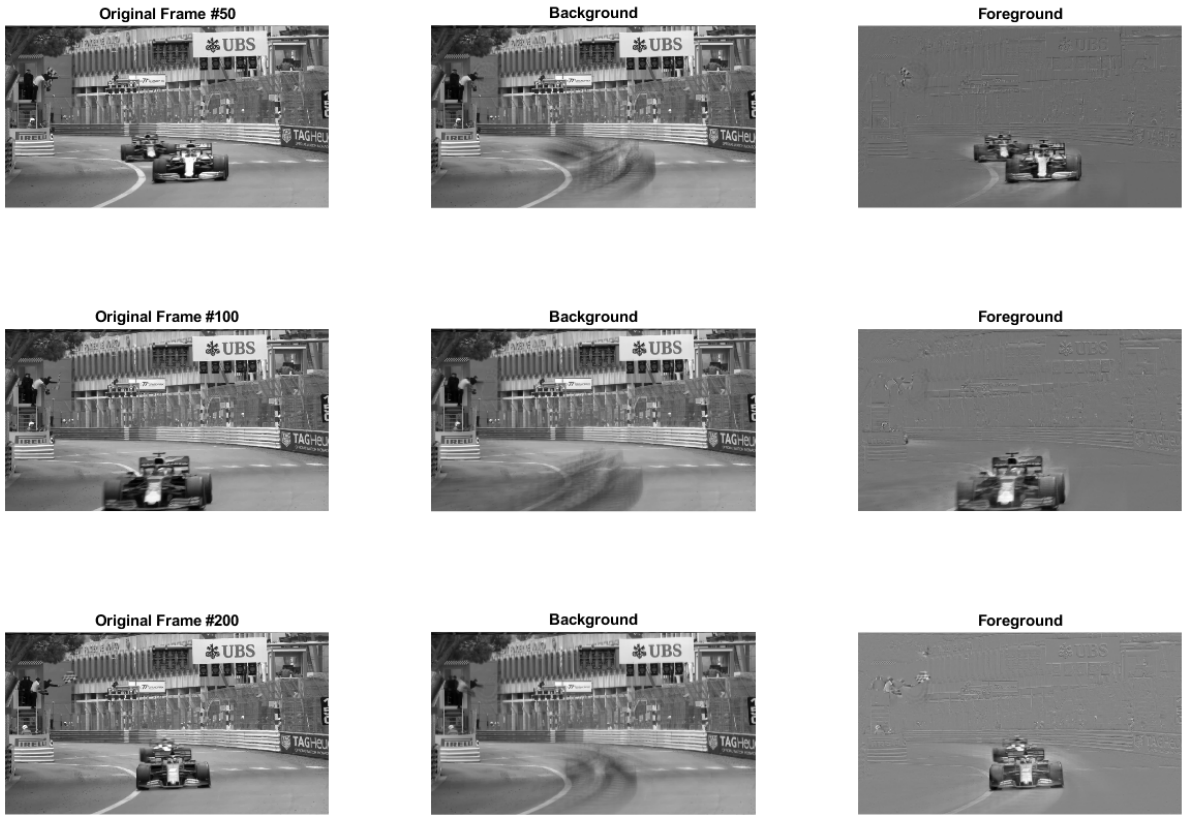

Figure 3: DMD applied to frame 50, 100, and 200 of the racecar video.

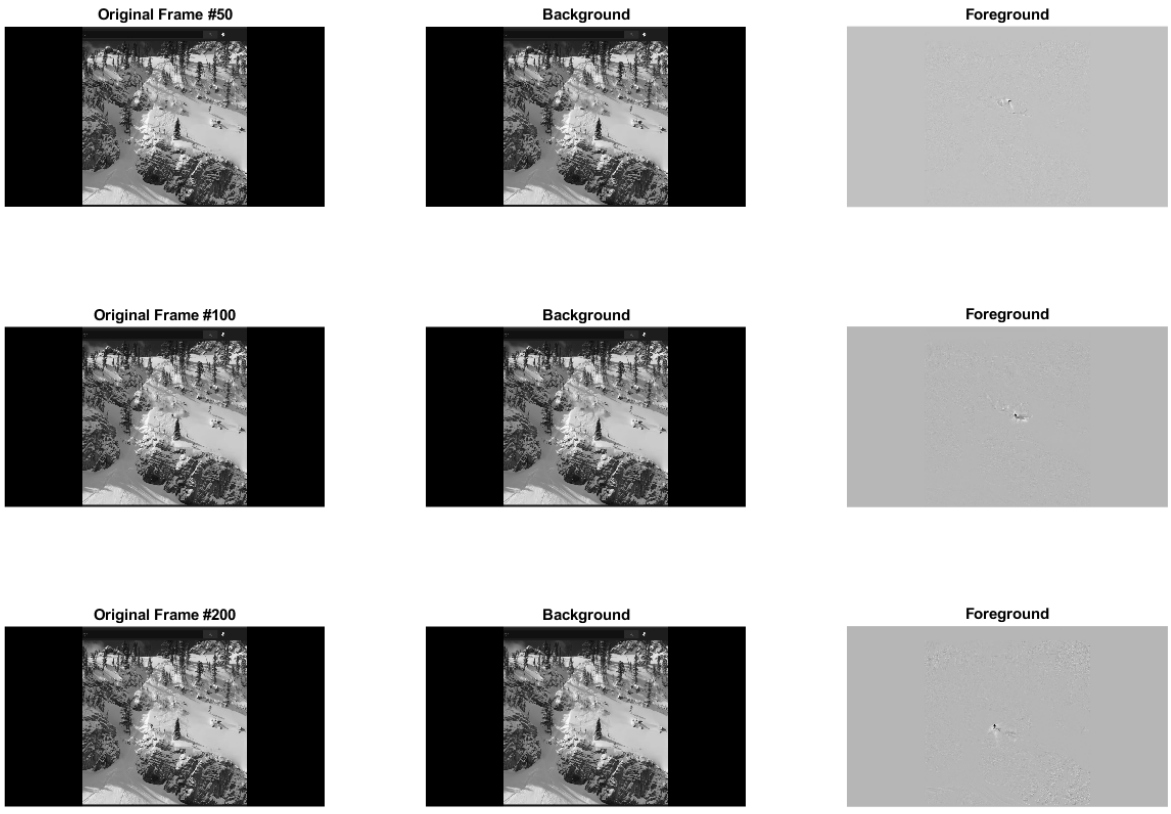

Figure 4: DMD applied to frame 50, 100, and 200 of the ski video.

5 Summary and Conclusions

Tracking moving objects is an important tool for many computer vision algorithms and many background subtraction methods have been developed to solve this problem. The simplest method, using the time-averaged background, requires "training" data with minimal foreground features to improve accuracy. The DMD method is an improvement over averaging methods because it can be applied to a video without training images and still recover some longer time scale movements from non-zero ω terms. The two videos tested with this algorithm had very good results, recovering clear videos of the moving objects with minimal projection of the background image. One of the downsides of this method was the high memory usage and longer runtimes, which was improved by the rank cutoff but still involved large matrix operations.

There are many newer background subtraction techniques that can process data relatively quickly, especially with the recent advances in convolutional neural networks. Despite these advances, one of the most common algorithms used is the Gaussian background mixture model, which is implemented by OpenCV as a fast, realtime background subtraction algorithm [3]. For this algorithm, each pixel is modeled using multiple Gaussian distributions and a threshold, resulting in a black and white image corresponding to objects in the foreground vs. background. Though the resulting image separation is not smooth like the DMD algorithm implemented here, the black and white image can be easily used for object tracking algorithms.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[2] Edward Anderson et al. *LAPACK users' guide*. SIAM, 1999.

[3] Pakorn KaewTraKulPong and Richard Bowden. "An improved adaptive background mixture model for real-time tracking with shadow detection". In: *Video-based surveillance systems*. Springer, 2002, pp. 135–144.

Appendix A MATLAB Functions

The MATLAB functions used for these calculations are listed below:

- `im = im2gray(I)` converts RGB image `I` to greyscale image `im` with 8-bit unsigned integer element values (0-255).
- `[u, s, v] = svd(X)` returns the rotation matrices `u` and `v` and the diagonal matrix `s` as the singular value decomposition of `X` such that `X=u*s*v'`
- `[V,D] = eigs(A, n, 'lm');` returns the first `n` eigenvalues as diagonal matrix `D` and first `n` eigenvectors as columns of `V`.
- `m = diag(M)` returns the diagonal vector `m` of matrix `M`
- `B = reshape(A, sz1, ... , szn)` reshapes the matrix `A` into an `sz1*...*szn` matrix
- `bool = ismember(v, group)` returns a boolean vector `bool` with the same dimensions `v` set to 1 if the element is a part of `group` and 0 if not.
- `v = VideoReader(fname)` creates a `VideoReader` object from the file `fname`.
- `frames = read(v)` outputs 4D RGB video matrix `frames` from `VideoReader` object `v`.
- `v = VideoWriter(fname)` creates a `VideoWriter` object from the file `fname`, must use `open()` and `close()` to start and stop writing.
- `writeVideo(v, im)` writes image frame `im` to `VideoWriter` object `v`.

Functions used for basic arithmetic or plot formatting are not included above, but the important figures are included in the Computation Results section above.

Appendix B MATLAB Code

All code is located in the "Homework 5" folder of the github repository (`https://github.com/wliverno/AMATH582`).

The main file for this project, `HW5.m`, calls the `dmdBgSub()` method and writes the videos files as shown below:

```
clear all, close all, clc;

[fgmc, bgmc] = dmdBgSub('monte_carlo_low.mp4', 100, 0.2);
n = size(fgmc, 3);
fgmcvid = VideoWriter('monte_carlo_FG');
bgmcvid = VideoWriter('monte_carlo_BG');
open(fgmcvid);
open(bgmcvid);
for i=1:n
    writeVideo(fgmcvid, fgmc(:, :, i));
    writeVideo(bgmcvid, bgmc(:, :, i));
end
close(fgmcvid);
close(bgmcvid);

[fgski, bgski] = dmdBgSub('ski_drop_low.mp4', 50, 0.2);
n = size(fgski, 3);
fgskivid = VideoWriter('ski_drop_FG');
bgskivid = VideoWriter('ski_drop_BG');
open(fgskivid);
open(bgskivid);
for i=1:n
    writeVideo(fgskivid, fgski(:, :, i));
    writeVideo(bgskivid, bgski(:, :, i));
end
close(fgskivid);
close(bgskivid);
```

The file `dmdBgSub.m` contains the full DMD background subtraction algorithm, outputting the separated videos and creating some visualizations for debugging:

```
function [fgVid, bgVid] = dmdBgSum(filename, rank, cutoff)
    frames = read(VideoReader(filename));
    [lx,ly,~,n] = size(frames)

    X = zeros(lx*ly, n);
    for i=1:n
        frame = double(im2gray(frames(:,:,:,i)));
        X(:, i) = reshape(frame, lx*ly, 1);
    end

    X1 = X(:, 1:end-1);
    X2 = X(:, 2:end);
```

```
% SVD of original image matrix, with rank cutoff applied
%rank=100;
[U, S, V] = svd(X1, 'econ');
U = U(:, 1:rank);
V = V(:, 1:rank);
S = S(1:rank, 1:rank);

% DMD calculations applying Koopman operator to generate dynamic modes
Ar = U'*X2*V/S;
[W, D] = eig(Ar);
Lambda = diag(D);
omega = log(Lambda);
phi = U*W;
x0 = phi\X(:,1);
u_modes = zeros(rank,n);
for i = 1:n
  u_modes(:,i) =(x0.*exp(omega*i));
end

% Plot calculated omega values in the complex plane
figure; plot(real(omega), imag(omega), 'ko');
title('DMD mode \omega-values'), xlabel('Re(\omega)'), ylabel('Im(\omega)');

% Reconstruct background using omega cutoff
modes = abs(omega)<cutoff;
u_back = phi(:, modes)*u_modes(modes, :);

% Specify which frames to preview
figure;
preview = [50, 100, 200];
% Write background and foreground images frame by frame
j = 1;
fgVid = uint8(zeros(lx, ly, n));
bgVid = uint8(zeros(lx, ly, n));
for i=1:n
    backgroundvec = abs(u_back(:,i));
    framevec = abs(X(:,i));
    foregroundvec = framevec-backgroundvec;
    % negInds = foregroundvec<0;
    % backgroundvec(negInds) = backgroundvec(negInds) - foregroundvec(negInds);
    foregroundvec = foregroundvec - min(foregroundvec);
    bg = uint8(reshape(backgroundvec, lx, ly)*255/max(backgroundvec));
    fg = uint8(reshape(foregroundvec, lx, ly)*255/max(foregroundvec));
    % Plot the original, FG, and BG of specified frames
    if ismember(i, preview)
        im = uint8(reshape(framevec, lx, ly)*255/max(framevec));
        subplot(length(preview),3,(j-1)*3+1), imshow(im), title(['Original Frame #',num2str(preview
        subplot(length(preview),3,(j-1)*3+2), imshow(bg), title('Background'), drawnow;
        subplot(length(preview),3,(j-1)*3+3), imshow(fg), title('Foreground'), drawnow;
        j=j+1;
    end
    bgVid(:, :, i)=bg;
    fgVid(:, :, i)=fg;
end
```

```
end
```
